



Leseprobe

Thorsten Kansy

Datenbankprogrammierung mit .NET 4.0

Herausgegeben von Dr. Holger Schwichtenberg

ISBN: 978-3-446-42120-2

Weitere Informationen oder Bestellungen unter

<http://www.hanser.de/978-3-446-42120-2>

sowie im Buchhandel.

### 2.1.3 Vereinfachte Berechnung

Ebenfalls klein, aber fein ist die aus C# bekannte Möglichkeit, Variablen vereinfacht um einen bestimmten Wert zu verändern. Passend zu den Variablen @Name und @ID von oben sieht dies so aus:

```
SET @Name += ', Thorsten';
```

```
SET @ID -= 100;
```

Dies funktioniert auch mit allen anderen geeigneten Operatoren wie z.B. den Grundrechenarten etc.

### 2.1.4 FileStream-Storage

Mit SQL Server 2008 eröffnet sich die Möglichkeit, (binäre) Daten getrennt von den restlichen Daten der Tabelle in einem FileStream-Storage anzulegen. Die entsprechenden Inhalte einer Spalte werden dann nicht in den Datenmedien der Datenbank abgelegt, sondern direkt im Dateisystem, das dafür zwingend mit NTFS<sup>1</sup> formatiert worden sein muss. In der Tabelle befinden sich dann quasi nur noch Verweise. Somit können effizient sehr große Datenmengen gespeichert werden. Der SQL Server legt diese BLOB-Daten selbstständig im NTFS-Dateisystem ab und sorgt gleichzeitig bei Änderungen für die notwendigen Unterstützungen bei Transaktionen, Sicherungen/Wiederherstellung und Replikationen. Unter 3.11, „*Filestream*“, finden Sie beschrieben, wie mit diesem neuen Feature gearbeitet werden kann.

### 2.1.5 CDC (Change Data Capture)

Mit CDC können auf einfache Weise Änderungen am Inhalt und am Schema einer Tabelle überwacht und in eine Historientabelle geschrieben werden.

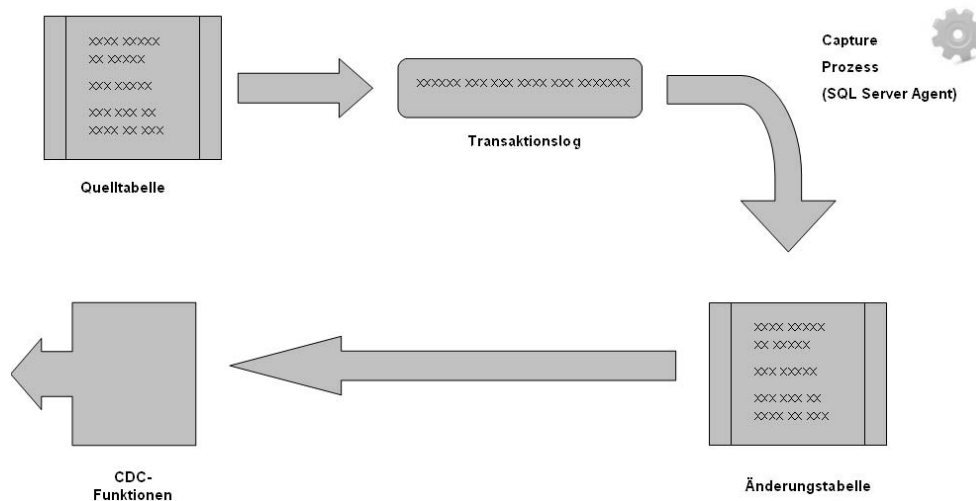
**Hinweis:**

Unter 2.1.6, „Change Tracking“, ist auf den ersten Blick ein recht ähnliches Feature beschrieben. Zur Orientierung finden Sie dort auch eine Entscheidungshilfe, wann CDC und wann Change Tracking die bessere Wahl ist.

Anders als bisherige Lösungen arbeitet CDC nicht mit Triggern, sondern als Auftrag des SQL Server-Agenten, der das Transaktionsprotokoll für diesen Zweck nutzt. Der Vorteil liegt dabei auf der Hand: Neben der besseren Performance besteht nicht die Gefahr, dass ein Fehler beim Protokollieren beeinträchtigt, wie es bei Triggern der Fall war.

---

<sup>1</sup> NTFS: New Technology File System, das Standarddateisystem für Windows XP, Windows Vista und Windows Server



**Abbildung 2.2** Diese Abbildung zeigt die Funktionsweise von CDC.

**Hinweis:**  
Details darüber, wie CDC eingesetzt werden kann, finden Sie unter Abschnitt 3.7, „CDC (Change Data Capture)“, der sich vollständig dieser Technik widmet.

### 2.1.6 Change Tracking

Werden Änderungen an Daten mittels Change Tracking verfolgt, so geschieht dies im Vergleich zu CDC nicht asynchron, sondern zeitgleich mit der Änderungsanweisung (INSERT, DELETE, UPDATE und MERGE, nicht jedoch TRUNCATE TABLE). Das bedeutet, dass Change Tracking die Geschwindigkeit dieser Anweisungen abbremst, dafür die Änderungen jedoch umgehend abgerufen werden können — eine Verzögerung, die, wie bei CDC, abhängig von der Auslastung des Systems ist, muss nicht berücksichtigt werden.

Change Tracking arbeitet mit Versionsnummern, die mit 0 beginnen und suggestiv erhöht werden, wenn Änderungen durchgeführt werden. Diese Versionsnummern sind für die gesamte Datenbank gültig und stellen damit die Grundlage für eine effiziente Datenreplikation dar, wie sie z.B. auch vom Microsoft Sync Framework realisiert wird, um zwei Datenbestände zu replizieren.

Diese funktioniert in Kürze so: Seite A hat eine Tabelle, von der Seite B weiß, dass sie bis zur Änderungsversion N auf dem aktuellstem Stand ist. Verbinden sich nun beide Systeme miteinander, kann Seite B abfragen, welche Versionsnummer gerade aus Seite A die aktuelle ist. Ist dieser Wert gleich N, so wurde in der Zwischenzeit nichts verändert. Ist der Wert allerdings größer N, so kann Seite B fragen „Gibt mir alle Änderungen der Tabelle in Version N“, den eigenen Datenbestand entsprechend auffrischen und schließlich N auf den neuen, höheren Wert aktualisieren. Später wiederholt sich der Prozess, um dann die neuen

Änderungen an den Daten erneut zu replizieren, usw. usw. Ist der Wert jedoch kleiner, so liegt ein Problem vor, und die beiden Seiten A & B müssen sich komplett neu initialisieren — oftmals so, dass eine Seite alle Daten der anderen übernimmt. Was dieses System nicht hergibt, ist der genaue Zeitpunkt, wann eine Änderung vorgenommen wurde.

Informationen über Änderungen sind nur für einen definierbaren Zeitraum verfügbar. Wird versucht, nach dessen Ablauf Änderungen abzufragen, sind diese nicht mehr gültig oder lückenhaft. Wie Sie nach der Aktivierung weiter unten sehen werden, gibt es jedoch einen Mechanismus, der solche Fälle erkennt und angemessen reagiert.

**Hinweis:**

Details darüber, wie Change Tracking in der eigenen Anwendung eingesetzt werden kann, finden Sie unter Abschnitt 3.8, „Change Tracking“.

### CDC vs. CT

Die Funktionalitäten von CDC und CT erscheinen recht ähnlich, wenn nicht sogar gleich. Doch es gibt Unterschiede, die wichtig für die Entscheidungsfindung sind, wenn es darum geht, welche der beiden die richtige Wahl ist. Die folgende Tabelle stellt die unterschiedlichen Merkmale gegenüber.

**Tabelle 2.1** Gegenüberstellung der Merkmale von CDC und CT

Merkmals	CDC	CT
Asynchrone Ausführung	Ja	Nein
Zeitgleiche, synchrone Ausführung	Nein	Ja
Benötigt SQL Server Agent	Ja	Nein
Erfasst Nettodatenänderungen	Ja	Ja
Erfasst Bruttodatenänderungen	Ja	Nein
Tabellengenaue Überwachung	Ja	Ja
Spaltengenaue Überwachung	Ja	Ja
Art der Änderung ist bekannt	Ja	Ja
Genauer Zeitpunkt ist bekannt	Ja	Nein
Änderungskontext	Nein	Ja
Erfasst Änderungen am Aufbau von Tabellen	Ja	Nein
Erfasst jegliche Änderungen an und in der Datenbank, inklusive Leseoperation auf einer Tabelle, und hält zudem fest, wer für die Änderung verantwortlich ist.	Nein	Nein

Das letzte Merkmal der vorherigen Tabelle, nämlich das Erfassen von Leseoperationen in der Datenbank, ist weder mit CDC noch mit CT möglich. Für Anwendungen oder Situationen, die solch eine Funktion benötigen, sei auf Auditing hingewiesen, das im nächsten Abschnitt beschrieben wird.

### 2.1.7 Auditing

Mit Auditing lassen sich ab SQL Server 2008 alle Arten von Zugriffen auf Datenbanken aufzeichnen. Da es sich hierbei um ein Feature handelt, das zur vollständigen, jedoch selektiven Überwachung gedacht ist, kann über eine Spezifikation festgelegt werden, welche Aktion (z.B. DML-Anweisungen) von welchen Benutzern/Rollen dabei berücksichtigt wird.

Auditing bedient sich der erweiterten Ereignisse (Extended Events), der neuen Ereignis-Infrastruktur, die mit SQL Server 2008 eingeführt wurde. Der praktische Einsatz gliedert sich grob in die vier folgenden Schritte aus:

- Die Erstellung und das Aktivieren eines Überwachungsobjekts auf Instanzebene, das alle Zugriffe zu einem Ziel sendet. Dies kann eine Datei, das Anwendungs- oder das Sicherheitsprotokoll des Betriebssystems sein.
- In den betreffenden Datenbanken kann eine Überwachungsspezifikation angelegt werden, die festlegt, welche Aktionen von wem in dieser Datenbank aufgezeichnet werden sollen.
- Aktivierung der Überwachung
- Später Auswertungen und Analyse der Zugriffe

SQL Server Auditing ist ein Feature, das für die Überwachung und den Nachweis des Zugriffs auf sensible Dateien entwickelt wurde. Daher kann es vollständig per Oberfläche administriert und die Protokollierung außerhalb der Datenbank abgelegt werden — damit ist z.B. verstärkt sichergestellt, dass der Zugriff auf diese empfindlichen Daten wirklich nur von einer auserlesenen Schar von Anwendern (Revision) möglich ist.

Jedoch lassen sich die gewonnenen Informationen leicht per T-SQL abfragen, sodass auch ein Einsatz in der eigenen Anwendung keine großen Probleme bereitet.

**Hinweis:**

Eine kleine Einführung, wie Auditing aktiviert und verwendet werden kann, finden Sie in Kapitel 3 unter 3.9, „Auditing“.

### 2.1.8 Die Merge-Anweisung

Neu in der Familie der DML-Anweisungen ist die ANSI<sup>2</sup>-SQL-konforme MERGE-Anweisung, die INSERT-, DELETE- und UPDATE-Anweisungen in sich vereint. So ist es z.B. möglich, mit nur einer Anweisung Zeilen in einer Tabelle entweder zu aktualisieren oder einzufügen, je nachdem, ob die Zeile schon vorher existierte. Dabei eignet sich diese Anweisung besonders in Data-Warehousing-Szenarien.

---

<sup>2</sup> ANSI: American National Standards Institute

Ohne die MERGE-Anweisung könnte eine solche Aufgabenstellung mit einer entsprechenden Kombination aus INSERT-, DELETE- und UPDATE-Anweisungen erledigt werden. Diese haben aber klar den Nachteil, dass Daten mehrfach abgefragt werden müssen und, da es sich nicht um eine einzelne Anweisung handelt, Vorkehrungen für eine Transaktion getroffen werden müssen — ein Umstand, der sich gerade bei Abfragen, die auf mehrere Server verteilt wurden, als langsam und kompliziert erweist.

```
-- Zuerst das Ziel festlegen (Tabelle oder Sicht)
MERGE Ziel Z
-- Nun die Quelle festlegen (Tabelle, Sicht oder Abfrage)
USING Quelle Q
-- Verbindung zwischen Zeilen im Ziel und in der Quelle
ON Q.ID = Z.ID
-- Zeile aktualisieren, wenn in Quelle und Ziel vorhanden
WHEN MATCHED THEN UPDATE SET Name = Q.Name
-- Zeile einfügen, wenn in Quelle, aber nicht im Ziel vorhanden
WHEN NOT MATCHED THEN INSERT (id, Name) VALUES (id, Name)
-- Zeile löschen, wenn im Ziel, aber nicht (mehr) in der Quelle vorhanden
WHEN NOT MATCHED BY SOURCE THEN DELETE
--Zurückgeben, was mit welcher Zeile geschehen ist
OUTPUT $ACTION, Q.ID;
```

Über das Pseudonym \$ACTION lassen sich zudem in Verbindung mit einer OUTPUT-Klausel die Veränderungen für jede Zeile abrufen (INSERT; DELETE oder UPDATE).

#### Hinweis:

Unter 4.5.5, „Die Merge-Anweisung“, finden Sie weitere Details zu dieser neuen Anweisung.

### 2.1.9 Mehrere Zeilen mit einer INSERT-Anweisung einfügen

Bis SQL Server 2005 konnten pro INSERT-Anweisung nur jeweils die Werte für eine einzige Zeile angegeben werden. Mehr als eine Zeile konnte damit nicht eingefügt werden, sodass eine Reihe von „Notlösungen“ existierten, die z.B. mit SELECT und dem UNION ALL-Operator als Wertequelle arbeitet. Nun ist dies nicht mehr notwendig. Die INSERT-Anweisung akzeptiert nun Werte einer beliebigen Anzahl Zeilen:

```
INSERT INTO Tabelle VALUES
('Kansy', 'Thorsten', DEFAULT), -- Zeile 1
('Bond', 'James', 54),         -- Zeile 2
('Sponge', 'Bob', DEFAULT);    -- Zeile 3
```

Können die Werte einer Zeile nicht eingefügt werden, so wird die gesamte Anweisung abgebrochen und kein Wert eingefügt. Schließlich ist die gesamte Anweisung nach wie vor nur eine einzige.

### 2.1.10 Grouping Sets

Mit den neuen Grouping Sets lässt sich die Ausgabe einer SELECT-Anweisung nach mehr als nur einem Satz Spalten gruppieren. Dadurch lassen sich u.a. ähnliche, jedoch flexiblere Abfragen, wie sie mit WITH ROLLUP und CUBE möglich sind, realisieren.

```
SELECT Mitarbeiter, Jahr, SUM(Umsatz) AS Umsatz
FROM Umsaetze
GROUP BY GROUPING SETS
(
    (Mitarbeiter), -- Gruppierungssatz 1
    (Jahr)         -- Gruppierungssatz 2
);
```

Die Ausgabe kann so aussehen:

Mitarbeiter	Jahr	Umsatz
NULL	2005	37000.00
NULL	2006	44000.00
NULL	2007	39000.00
TKA	NULL	32000.00
HUT	NULL	34000.00
UNE	NULL	54000.00

Da die gleiche `SELECT`-Anweisung quasi mehrfach hintereinander gruppiert wird, gibt es bei Kombinationen, in denen eine Spalte in einer einzelnen Gruppierung nicht zulässig wäre, lediglich den Wert `NULL` als Rückgabe.

### 2.1.11 Tabellen als Parameter (Table-Valued Parameters – TVP)

Mit SQL Server 2008 können Funktionen und gespeicherte Prozeduren Parameter erhalten, die einen tabellarischen Aufbau haben (Table-Valued Parameters). Damit können statt einzelner Werte ganze Datenmengen übergeben werden. Das folgende Beispiel zeigt, wie dies in T-SQL realisiert wird.

```
-- Datentyp erzeugen
CREATE TYPE tParam AS TABLE
(
    id INT
);
GO

-- Gespeicherte Prozedur erstellen
CREATE PROC TestTVP
(
    @IDs tParam READONLY
)
AS
    -- Ausgabe zu Testzwecken
    SELECT * FROM @IDs;
GO

-- Aufruf
DECLARE @IDs tParam;
INSERT INTO @IDs VALUES (0815), (4711);

EXEC TestTVP @IDs;
```

Die Ausgabe sieht wie erwartet so aus:

id
815
4711

## 10 ADO.NET Entity Framework 4.0

Seit .NET Framework 3.5 Service Pack 1 bietet Microsoft einen eigenen O/R-Mapper<sup>1</sup> an: das ADO.NET Entity Framework (EF).

Mit .NET Framework 4.0 ist nun die überarbeitete und erweiterte Version mit grafischem Designer verfügbar. EF kann als technische Weiterentwicklung zu LINQ to SQL gesehen werden. Aus diesem Grund wird dem ADO.NET Entity Framework auch ein eigenes Kapitel in diesem Buch gewidmet, das als Einführung in diese Thematik gedacht ist.

Während LINQ to SQL noch recht eng mit dem SQL Server als Schicht zur Datenspeicherung verwoben ist, ist dies beim ADO.NET Entity Framework nicht mehr der Fall. Die Datenbank im Hintergrund ist stark abstrahiert, und es ist denkbar, Anwendungen zu schreiben, die völlig unabhängig von der verwendeten Datenbanktechnologie sind.

Das ADO.NET Entity Framework wurde bei seiner Entwicklung so konzipiert, dass Entwickler Datenzugriffsanwendungen erstellen können, indem sie für ein konzeptionelles Anwendungsmodell und nicht für ein relationales Datenbankschema programmieren. Das Ziel war es dabei auch, die Menge des Codes und den Wartungsaufwand zu verringern, die für datenorientierte Anwendungen erforderlich sind.

**Wichtig:**

Dieses Kapitel ist als Einführung in das ADO.NET Entity Framework 4.0 gedacht. Es soll dem geneigten Leser, der sich bereits mit SQL Server, der T-SQL-Abfragesprache und ADO.NET vertraut gemacht hat, helfen, sich in dieser Technologie zurechtzufinden.

---

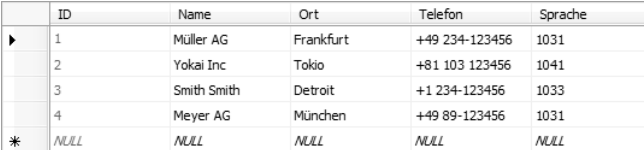
<sup>1</sup> O/R-Mapper dienen zur objektrelationalen Abbildung und sind damit das Bindeglied zwischen Klassen auf .NET-Seite und Tabellen auf Seiten der Datenbank.

## 10.1 Wozu dient ein O/R-Mapper?

Gerade am Anfang stellt sich womöglich die Frage, warum man überhaupt einen O/R-Mapper verwenden soll; schließlich gibt es eine ungeheure Menge von Programmen auf diesem Planeten, die dies nicht tun und trotzdem gut laufen. Der Hauptgrund ist einfach: schnellere Entwicklung mit weniger Fehlerpotenzial, da es so einfach möglich ist, auf Spalten einer Tabellenzeile als Eigenschaft einer Klasse zuzugreifen. Die gesamte Tabelle wird als Auflistung von Instanzen dieser Klassen dargestellt. Auf diese Weise ist es mit Unterstützung der IntelliSense-Funktion des Editors möglich, sich ein wenig bei der Quellcodeerstellung unterstützen zu lassen. Zudem können auch Abfragen mit LINQ to Entities verwendet werden.

```
Kunden Kunde = KundenObjectContext.Kunden
    .Where(k => k.Name == "Müller AG").FirstOrDefault();
```

Existiert z.B. in einer SQL Server-Datenbank folgende Tabelle mit dem Namen „Kunden“,



	ID	Name	Ort	Telefon	Sprache
▶	1	Müller AG	Frankfurt	+49 234-123456	1031
	2	Yokai Inc	Tokio	+81 103 123456	1041
	3	Smith Smith	Detroit	+1 234-123456	1033
	4	Meyer AG	München	+49 89-123456	1031
*	NULL	NULL	NULL	NULL	NULL

Abbildung 10.1 Die Tabelle mit einigen Zeilen auf dem SQL Server

so wird diese (vereinfachte) Klasse für die Daten einer Zeile generiert.

```
public partial class Kunden
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string Ort { get; set; }
    public string Telefon { get; set; }
    ...
}
```

Auf diese Weise kann auf jede „Spalte“ wie gewohnt als Eigenschaft zugegriffen werden (lesend und schreibend).

```
// Instanz wird über das ADO.NET Entity Framework erzeugt
Kunde meinKunde = ...;

meinKunde.Name = "Müller AG";
meinKunde.Stadt = "München";
```

Die gesamte Tabelle wird von einer Auflistung (ObjectSet) repräsentiert und ist über eine besondere Klasse, den Objektcontext (siehe 10.7.1.2, „Der Objektcontext“), verfügbar.

```
public ObjectSet<Kunden> Kunden
{
    get ...;
}
```

Das Erstellen dieser Klassen wird über den entsprechenden Generator des Visual Studios erzeugt. Das Laden/Speichern der Daten aus der Datenbank geschieht dann über das ADO.NET Entity Framework 4.0.

Doch vor der Verwendung muss das Framework erst zum Einsatz gebracht werden.

## 10.2 ADO.NET Entity Framework im Projekt

---

Es gibt zwei Wege, das ADO.NET Entity Framework zu verwenden. Der erste besteht darin, dem Projekt ein entsprechendes Framework hinzuzufügen. Dies geschieht dann mit dem grafischen Designer beim Entwurf der gewünschten Entitäten und kann als Vorlage eine bestehende Datenbank verwenden. Der andere kommt ganz ohne Assistent aus und nennt sich daher „Code-Only“-Ansatz. Er ermöglicht es, bestehende Klassen in der Datenbank zu persistieren (siehe 10.7, „Code-Only“).

### 10.2.1 Entitätenmodell (EDM)

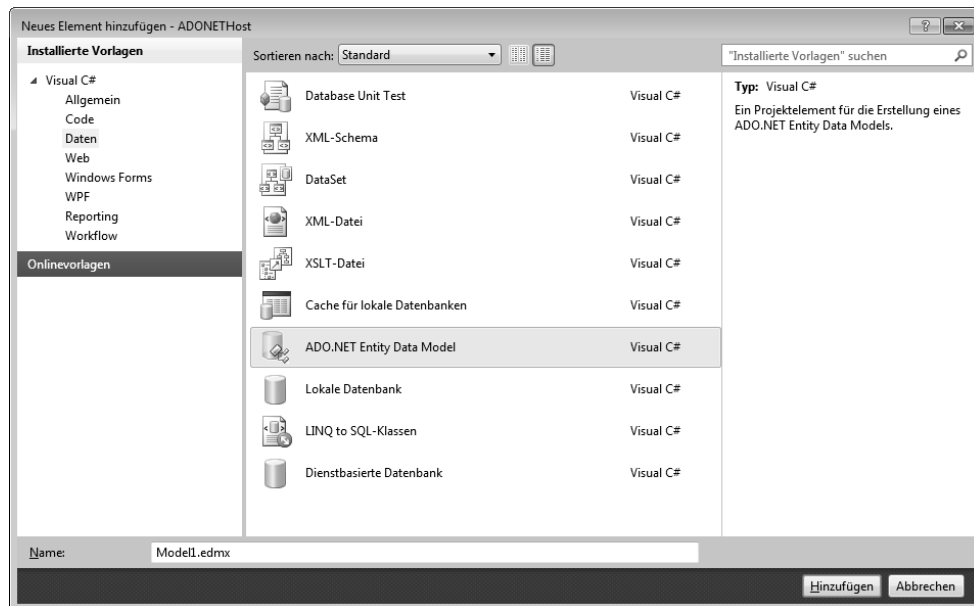
In den folgenden Schritten wird beschrieben, wie mithilfe von Vorlagen und Assistenten ein Entitätenmodell erzeugt und dem aktuellen Projekt hinzugefügt werden kann. Das Entitätenmodell stellt die notwendigen Metadaten dar, damit die Entitäten, deren Beziehungen und weitere relevanten Informationen vorhanden sind.

Grob stehen dafür zwei unterschiedliche Ansätze zur Auswahl: Entweder die Datenbank liegt schon in fertiger Form vor, und es soll nach deren Vorbild ein Entitätenmodell erzeugt werden („Database first“), oder es wird der umgekehrte Weg beschritten, und das Entitätenmodell wird erst erstellt und anschließend auf dem SQL Server angelegt („Model first“). Im Folgenden werden beide Wege der Reihe nach aufgezeigt.

#### 10.2.1.1 Database first

Dieses Vorgehen ist hilfreich, wenn die Datenbank bereits besteht und ein passendes Entitätenmodell erzeugt werden soll.

1. Über das Kontextmenü fügen Sie dem Projekt ein Element „ADO.NET Entity Data Model“ hinzu.



**Abbildung 10.2** Dem Projekt muss das entsprechende Entitätenmodell hinzugefügt werden.

Ein solches besteht aus zwei Komponenten:

- eine partielle CS-Datei (wenn VB.Net verwendet wird, eine VB-Datei)
  - eine XML-Datei
2. Im darauffolgenden Schritt kann eine Datenbank als Vorlage für das Modell angelegt oder mit einem leeren Modell begonnen werden. Beides endet im grafischen Designer, der im 5. Schritt zu sehen ist.
  3. Wird die Datenbank als Vorlage ausgewählt, so gilt es im nächsten Dialog, die Datenbankverbindung auszuwählen bzw. anzulegen. Standardmäßig ist es möglich, die Verbindungsinformation unter dem angegebenen Namen in die Konfiguration der Anwendung aufzunehmen.

Die Verbindung zur Datenbank beschränkt sich dabei keinesfalls nur auf SQL Server!



Abbildung 10.3 Die Festlegung der Verbindung zu der Datenbank

4. Im nächsten und letzten Schritt kann bestimmt werden, was aus der Datenbank für das ADO.NET Entity Framework verwendet werden soll. Zur Auswahl stehen dafür bei der Verwendung eines SQL Servers:
- Tabellen
  - Sichten
  - Gespeicherte Prozeduren
  - Funktionen

Dabei ist es möglich, alle Tabellen, Sichten etc. für das Modell auszuwählen oder gezielt einzelne auszusuchen.

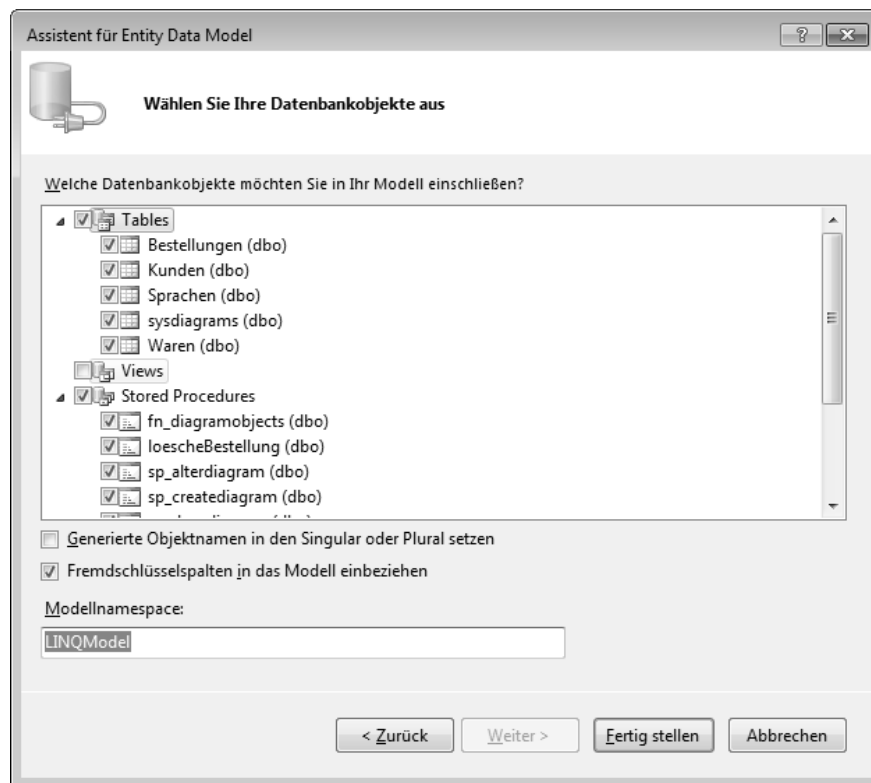


Abbildung 10.4 Bestimmung der Objekte für das Entitätenmodell

5. Nachdem der Assistent abgeschlossen wurde, ist das Modell fertig und kann verwendet oder weiter angepasst werden.

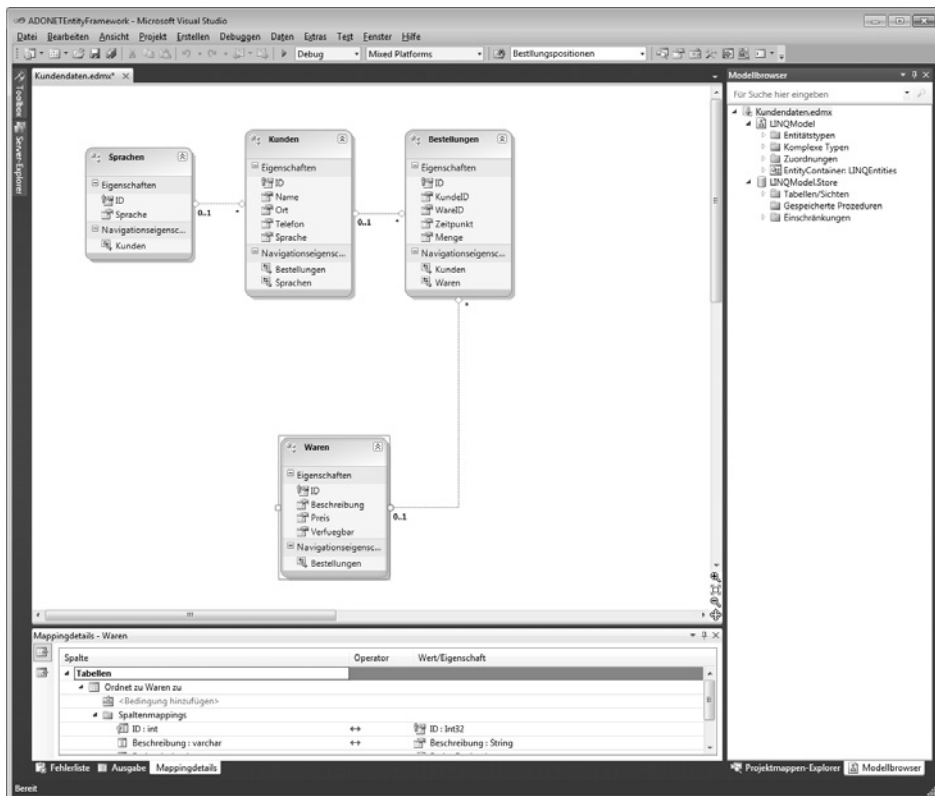


Abbildung 10.5 Der grafische Designer des Visual Studios für Entitätenmodelle

Ein Entitätenmodell lässt sich auch im Modellbrowser anzeigen. Dieser erscheint dort, wo auch der Projektmappen-Explorer standardmäßig zu finden ist (auf der rechten Seite des Visual Studios).

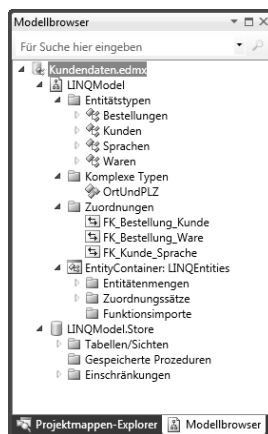


Abbildung 10.6 Die alternative Darstellung im Modellbrowser

### 10.2.1.2 Model first-Ansatz

Der „Model first“-Ansatz ist dann richtig, wenn noch keine entsprechende Datenbank existiert und das entsprechende Entitätenmodell per Hand erstellt werden muss.

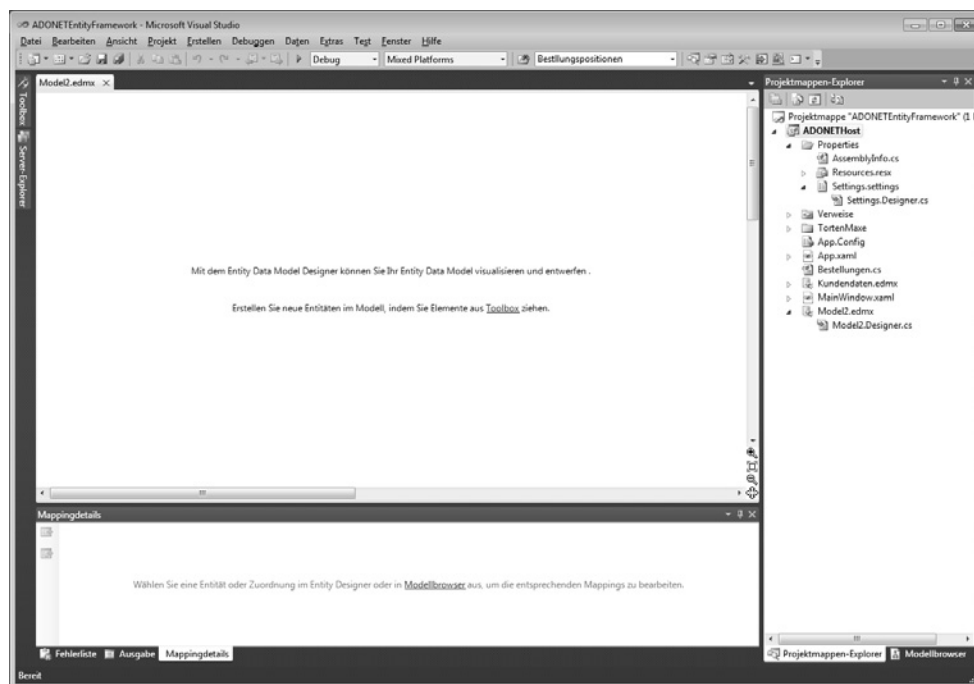
**Wichtig:**

Es wird nicht die Datenbank im eigentlichen Sinne erstellt (diese muss nämlich bereits existieren), sondern es werden die notwendigen Objekte in einer Datenbank erstellt. Dies sind vor allem Tabellen zum Speichern der Daten.

- Über das Kontextmenü fügen Sie dem Projekt ein Element „ADO.NET Entity Data Model“ hinzu. Dieser Schritt ist in „**Abbildung 10.2** Dem Projekt muss das entsprechende Entitätenmodell hinzugefügt werden“ zu sehen.

Ein solches besteht, obwohl anfangs leer, aus zwei Komponenten:

- eine partielle CS-Datei (wenn VB.Net verwendet wird, eine VB-Datei)
  - eine XML-Datei
- Im zweiten Schritt muss ein leeres Model ausgewählt werden, das in den nächsten Schritten aufgebaut wird.



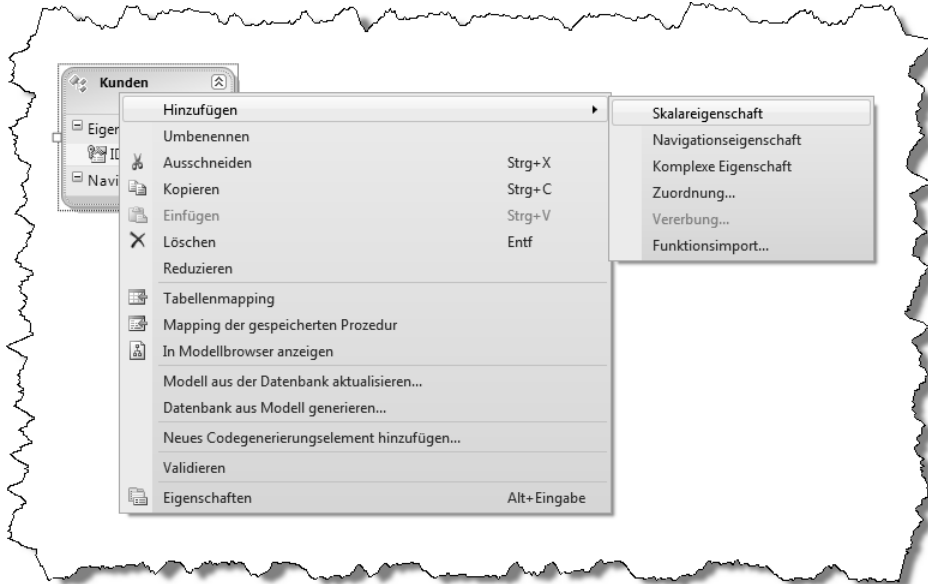
**Abbildung 10.7** Das jungfräuliche Entitätenmodell mit der Toolbox an der linken Seite

- Dem Entitätenmodell können per Drag&Drop weitere Entitäten hinzugefügt werden. Hier stehen Entitäten, Zuordnungen (Assoziationen) und Vererbungen zur Auswahl.

**Hinweis:**

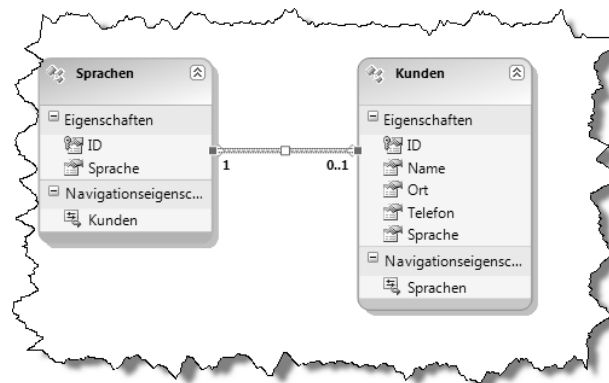
Vieles kann dem Entitätenmodell auch über dessen Kontextmenü (Rechtsklick auf den „leeren“ Bereich) hinzugefügt werden.

4. Jeder Entität können über den Designer, Skalareigenschaften (für einfache Werte), komplexe Eigenschaften (für zusammengesetzte Werte) etc. hinzugefügt werden.



**Abbildung 10.8** Entitäten können per Kontextmenü nach und nach aufgebaut werden.

5. Zuordnungen (also Beziehungen) zwischen Entitäten lassen sich durch Auswahl des entsprechenden Werkzeugs und anschließendes Auswählen der Entitäten im Designer festlegen.



**Abbildung 10.9** Die Beziehung zwischen zwei Entitäten